CSC449, HW#2, Kefu Zhu

1. Use of `dev` dataset

Since the I did not tune the learning rate in this assignment and I am using the SGD method to find the optimal weight vector, some updates during the interation can potentially overshoot the local minima of the cost function.

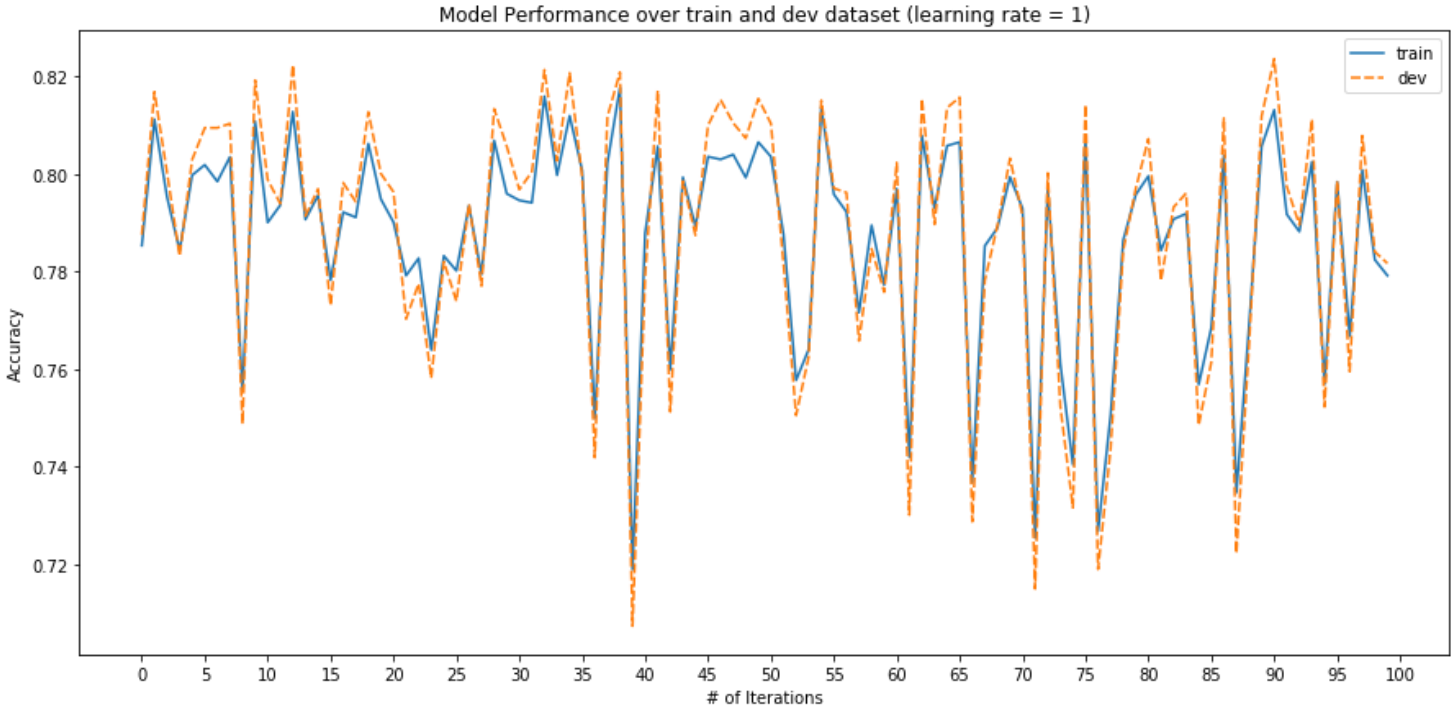
Therefore, I treat the weight vector (model) from each iteration as separate models and created a variable `best_weights` to keep track the best weight vector that has been computed so far based on the test performance on `dev` dataset.

Rather than outputing the weight vector that results from the last iteration, I will return the best weight vector among all iterations when `--nodev` is not provided.

2. Model Training

As shown in the graph below, we can clearly see the model accuracy on both the `train` dataset and the `dev` dataset flunctuate a lot through the iterations, which also proves that the update is clearly overshooting a lot during the interation

```
# Record the model performance (learning rate = 1)
performance_df_1 = model_performance(1,train_x,train_y,dev_x,dev_y)
# Plot the model performance (learning rate = 1)
fig, ax = plt.subplots(figsize = (15,7))
sns.lineplot(data = [performance_df_1.loc[:,'train_accuracy'],
                     performance_df_1.loc[:,'dev_accuracy']])
ax.set_title('Model Performance over train and dev dataset (learning rate = 1)');
ax.set_xlabel('# of Iterations');
ax.set_ylabel('Accuracy');
ax.set_xticks(range(0,101,5));
ax.legend(['train','dev'])
```



After changing the learning rate from 1 to 0.01, the problem of overshooting is still significant. However, if we pay attention to the y-axis, we are able to reach higher accuracy than before.

```
# Record the model performance (learning rate = 0.01)
performance_df_001 = model_performance(0.01,train_x,train_y,dev_x,dev_y)
# Plot the model performance (learning rate = 0.01)
fig, ax = plt.subplots(figsize = (15,7))
sns.lineplot(data = [performance_df_001.loc[:,'train_accuracy'],
                     performance_df_001.loc[:,'dev_accuracy']])
ax.set_title('Model Performance over train and dev dataset (learning rate = 0.01)'
ax.set_xlabel('# of Iterations');
ax.set_ylabel('Accuracy');
ax.set_xticks(range(0,101,5));
ax.legend(['train','dev'])
```

3. Appendix

`model_performance` : Function used to produce the model performance grpah

```
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt

def model_performance(learning_rate,train_xs,train_ys,dev_xs,dev_ys):

    # Initialize weight vector to be all zeros
    weights = np.zeros(NUM_FEATURES)
    # Initialize a dataframe to record model performance on train and dev dataset
    df = pd.DataFrame(columns=['iter_num','train_accuracy','dev_accuracy'], dtype=

    # Track the number of iterations
    num_iter = 0
    # Record the current number of iteration and model performance
    df.loc[num_iter,'iter_num'] = num_iter
    df.loc[num_iter,'train_accuracy'] = test_accuracy(weights, train_ys, train_xs)
    df.loc[num_iter,'dev_accuracy'] = test_accuracy(weights, dev_ys, dev_xs)
    # While the number of iteractions does not go beyond the maximum
    while num_iter < 100:
        # # Make a deep copy of weight vector before next iteration
```

```
        # old_weights = np.array([w for w in weights])
        # Loop through each pair of (x,y) in the training dataset
        for x,y in zip(train_xs,train_ys):
            # Classify current data point based on current weights
            y_hat = np.sign(np.dot(np.transpose(weights),x))
            # If we classify incorrectly
            if y_hat!=y:
                # Update the weight vector
                weights = np.add(weights,np.array(learning_rate*y*x))

        # Record the current number of iteration and model performance
        df.loc[num_iter,'iter_num'] = num_iter
        df.loc[num_iter,'train_accuracy'] = test_accuracy(weights, train_ys, train
        df.loc[num_iter,'dev_accuracy'] = test_accuracy(weights, dev_ys, dev_xs)

        # Increment the number of iteraction
        num_iter += 1
        # Print the progress
        if 100 <= 10: #
            print('# of iterations: {}'.format(num_iter))
        else:
            if num_iter % round(100/10) == 0:
                print('# of iterations: {}'.format(num_iter))

    return df
```